



Clase 9. Python

Funciones

***RECUERDA PONER A GRABAR LA
CLASE***





OBJETIVOS DE LA CLASE

- Crear funciones
- Retornar valores
- Enviar valores

CRONOGRAMA DEL CURSO

Clase 8



Manejo de archivos y datos



MI MASCOTA



CURIOSOS POR LA INFORMACIÓN

Clase 9



Funciones



PAR O IMPAR



FUNCIÓN AÑO BISIESTO

Clase 10



Funciones II



RELOJ



FUNCIONES

FUNCIONES



Funciones

Cuando creamos nuestros propios programas nos damos cuenta de que muchas de las tareas que implementamos se repiten o presentan de forma similar pero con algunos cambios.

Entonces aparece la necesidad de agrupar este código repetido o similar, a las agrupaciones de código se les denominan **funciones** las cuales se pueden ejecutar múltiples veces gracias a un nombre único que las identifica.



Funciones

Para comunicarse con nuestro proceso principal las funciones pueden recibir y devolver datos manipulados. Un ejemplo de una función que conocemos es `len()` que nos permite saber la cantidad de elementos de una colección.

*Recordemos que a esta función hay que pasarle el elemento del cual queremos saber la longitud y devuelve un valor entero con la longitud, a este valor se le denomina **valor de retorno**.*

```
>>> len("Hola")  
4
```

DEF



¿De qué se trata?

La sentencia **def** sirve para crear funciones definidas por el usuario.

Una definición de función es una sentencia ejecutable.



Sintaxis para una definición de función

```
>>> def NOMBRE(PARÁMETROS):  
        SENTENCIAS  
        RETURN [EXPRESIÓN]
```

- **NOMBRE:** Es el nombre de la función.
- **PARÁMETROS:** Como vimos en la clase 8 hay scripts con argumentos, en las funciones, cuando recibe argumentos se les denominan parámetros.



Sintaxis para una definición de función

- **SENTENCIAS:** Es el bloque de código.
- **RETURN:** Es una sentencia de Python, le indica a la función que devolver cuando llamemos a la función.
- **EXPRESIÓN:** Es lo que devuelve la sentencia `return`



Definir funciones básicas

```
>>> def saludar():  
    print("Estoy saludando desde la  
función")
```

La llamamos usando:

```
>>> saludar()
```



Definir funciones más avanzadas



```
>>> def saludar_con_nombre(nombre):  
    saludando = print("Hola {}! ¿Cómo  
estás?".format(nombre))  
    return saludando
```

Y la llamamos usando:

```
>>> saludar_con_nombre("Juan")
```



Recomendaciones

1. Utilizar minúsculas
2. Las palabras se separan con guiones bajos _
3. Utilizar nombres autoexplicativos
4. No usar nombres que no definan lo que hace la función (ejemplo letras simples o palabras sin sentido con lo que haga la función)

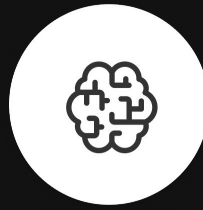
Variables y Funciones



```
>>> def test():  
    variable_test = 10  
    print(variable_test)  
  
>>> print(variable_test)  
NameError Traceback (most recent call last)  
<ipython-input-4-667d7c7a2c02> in  
<module>()  
----> 1 print(variable_test)
```

NameError: name 'variable_test' is not defined

Hay que tener en cuenta
que las variables creadas en
una función **no existen**
fuera de la misma.



¡PARA PENSAR!

Si las variables creadas en una función, sólo existen dentro de esa función ¿Cómo explicarías esto?

```
>>> variable_test = 10
>>> def test():
    print(variable_test)
>>> test()
```


CONTESTA EN EL CHAT DE ZOOM



Variables y Funciones

Sin embargo, hay que tener cuidado con las variables fuera de las funciones al usarlas en una función, ya que no puede llegar a funcionar como queremos:

```
>>> variable_test = 10
>>> def test():
    variable_test = 155
    print(variable_test)
>>> test()
```

El nt le da prioridad a la variable dentro de la función antes que a la de afuera.

RETORNANDO VALORES



Return



Las funciones pueden comunicarse con el exterior de las mismas, al proceso principal del programa usando la **sentencia return**. La comunicación con el exterior se hace **devolviendo valores**.

A continuación, un ejemplo de función usando return:

```
>>> def saludar_con_nombre(nombre):  
        saludando = print("Hola {}!  
        ¿Cómo estás?".format(nombre))  
        return saludando
```

Nota: Por defecto, las funciones retorna el valor None.

CODER HOUSE



Return



Sin embargo hay que tener en cuenta que la función **termina** al devolver un valor, es decir, lo que escribamos después **no** se ejecutará:

```
>>> def saludar_con_nombre(nombre):  
        saludando = print("Hola {}! ¿Cómo  
estás?".format(nombre))  
        return saludando  
        print("Hola mundo!")
```

Es similar a un break!



Return



```
>>> def numero():  
        return 6  
>>> s = numero() + 5  
>>> s  
11  
>>> a = numero() * 2 +  
5  
>>> a  
17
```

Los valores o variables retornados van a seguir siendo de un tipo de valor, por lo que podremos trabajarlos con lo que ya hemos visto



Return



Si vemos el tipo de dato de número, nos indicará que es un int:

```
>>> def numero():  
        return 6  
>>> type(numero())  
<class 'int'>
```

Por lo que no podremos sumar int a str aunque sea una función

```
>>> a = numero() + "hola"  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: unsupported operand type(s) for +: 'int' and  
'str'
```



Return



Algo interesante que pasa si devolvemos una colección es que **podemos utilizarla directamente desde la función** y hacer uso de las funciones internas de las colecciones:

```
>>> def lista():  
        return [1,2,3,4,5]  
>>> print(lista()[1:3])
```

Sin embargo, cada vez que hagamos un **print** a una función la estaremos llamando, por lo que **lo ideal es asignarlo a una variable y trabajarlo desde ahí:**

```
>>> variable = lista()  
>>> variable[1:4]
```



Return multiple



Una característica interesante, es la posibilidad de devolver valores múltiples separados por comas:

```
>>> def test():  
        return "Python", 20, [1,2,3]  
  
>>> test()  
( 'Python', 20, [1, 2, 3])
```


ENVIANDO VALORES



Enviando Valores a una función

Vimos como devolver valores y así comunicar una función con el exterior, ahora
enviar información desde el exterior a la función.

Para entender los conceptos más fácilmente vamos a trabajar alrededor de un
caso de estudio típico: Crear una función que sume dos números y retorne uno
en su resultado.



Enviando Valores a una función



Lo primero será definir una función la cual denominaremos como **suma** y recibirá 2 números con dos nombres como si fueran dos variables **numero1** y **numero2**, luego **retornamos la suma** entre ambos números.

```
>>> def suma(numero1, numero2):  
        return numero1 + numero2
```



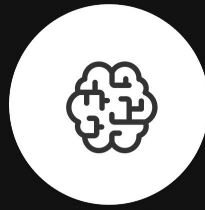
Enviando Valores a una función



Lo que hacemos para indicar que se reciben valores es **crear dos variables separadas por una coma**. Cuando nosotros llamemos a la función, automáticamente, **se le asignarán a estas variables los números que enviemos**, siguiendo el mismo orden:

```
>>> r = suma(7, 5)
```

En este caso 7 será la variable **numero1** y 5
será la variable **numero2**

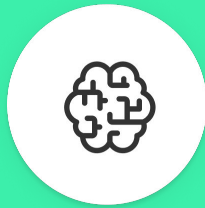


¡PARA PENSAR!

¿Qué ocurriría si lo hiciéramos al revés?

```
>>> r = suma(5, 7)
```

CONTESTA EN EL CHAT DE ZOOM



En este caso 5 será la variable **numero1** y 7 será la variable **numero2**. Hay que tener cuidado por como se pasan estos valores a la función, ya que si fuera otra operación matemática podría dar resultados muy distintos, como en una división o potencia.

Momentos de una función



Tenemos la **definición**

```
>>> def suma(numero1, numero2):  
    return numero1 + numero2
```

Y la **llamada**

```
>>> r = suma(7, 5)
```

¿Por qué es importante diferenciar?

Momentos de una función



Durante la definición de la función, las variables o valores se denominan **parámetros**:

```
>>> def suma(numero1, numero2):  
    return numero1 + numero2
```

Y durante la llamada se le denominan **argumentos**, como los argumentos de los scripts.

```
>>> r = suma(7, 5)
```




PAR O IMPAR

Realizar una función

Tiempo estimado: 3 minutos



DESAFÍO DE FUNCIONES CON PARÁMETROS

Tiempo estimado: 3 minutos

Realizar una función llamada par_o_impar:

1. Recibirá un número por parámetro
2. Imprimirá Par si el número es par
3. Imprimirá Impar si el número es impar
4. Si se ingresa algo que no sea número debe indicar que se ingrese un número. (Para los más audaces)



EJEMPLO SUBIDO AL DRIVE: [Desafío](#)

CODER HOUSE



FUNCIÓN AÑO BISIESTO

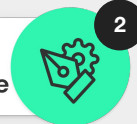
Crear una función con parámetros.

FUNCIÓN AÑO BISIESTO

Formato: El documento debe presentarse en Google Docs o mejor aún en Colabs, bajo el siguiente formato: `"FunciónAñoBisiesto+Apellido"`.

Sugerencia: En el formulario debe estar el print de pantalla de la consola con el ejercicio resuelto, como así también el código tipeado.

Desafío
entregable



>> Consigna: Realizar una función llamada `año_bisiesto`:

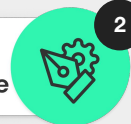
1. Recibirá un año por parámetro
2. Imprimirá "El año **año** es bisiesto" si el año es bisiesto
3. Imprimirá "El año **año** no es bisiesto" si el año no es bisiesto
4. Si se ingresa algo que no sea número debe indicar que se ingrese un número.

FUNCIÓN AÑO BISIESTO

Formato: El documento debe presentarse en Google Docs bajo el siguiente formato: "FunciónAñoBisiesto+Apellido". Colabs,

Sugerencia: En el formulario debe estar el print de pantalla de la consola con el ejercicio resuelto, como así también el código tipeado.

Desafío
entregable



>>Información a tener en cuenta al realizar el entregable:

Se recuerda que los años bisiestos son múltiplos de 4, pero los múltiplos de 100 no lo son, aunque los múltiplos de 400 sí. Estos son algunos ejemplos de posibles respuestas: 2012 es bisiesto, 2010 no es bisiesto, 2000 es bisiesto, 1900 no es bisiesto.

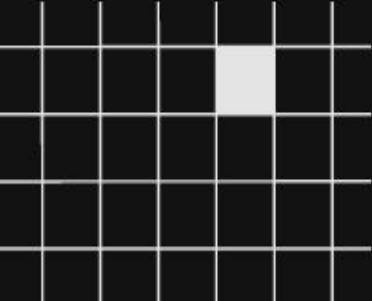
¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Funciones
 - Retorno de valores
 - Envío de valores
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE